

10
150 pts.

PRUN

Enciclopedia Práctica del Spectrum



Nueva Lente/Ingelek



EVALUADOR DE CADENAS



A creación de programas relativamente complejos, requiere del uso de las funciones BASIC. Ya en el último programa de ejemplo propuesto, hemos utilizado la función BASIC VAL, aunque sin describir sus características y modo de empleo.

De modo general, podemos decir que las funciones BASIC operan como «traductores», que a partir de un «vergúenza» que suministráis, predecen un valor de salida determinado. De esta forma, la estructura general de cualquier función es:

FUNCIÓN + ARGUMENTO

Como FUNCIÓN, se especifica cualquiera de las múltiples palabras clave de este tipo disponibles en el vocabulario BASIC, y como ARGUMENTO, encerrado entre paréntesis sólo cuando es estrictamente necesario, la expresión a evaluar, que puede ser tan complicada como se quiera, dependiendo de la clase de función elegida; la expresión ARGUMENTO será de tipo numérico o de cadena.

Lógicamente, los argumentos específicos para las diferentes funciones deben ser coherentes con la función escogida, produciéndose, en caso contrario, una detención del programa con el correspondiente mensaje de error. Este será el frecuente del tipo C Nonsense in BASIC, que podemos traducir al Castellano como C Binsensido en BASIC o C Tontearia en BASIC, si somos más coloquiales y duros con nosotros mismos. En

otras muchas ocasiones, los errores de este tipo con funciones, pueden ser ya detectados en la fase de corrección de sintaxis por ese sistema tan eficaz que ya conocemos: el syntax checker.

La utilización de una función siempre lleva consigo la presencia de un argumento.

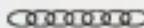
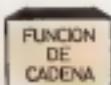


Por los resultados que podemos obtener de ellas, dividiremos las funciones en dos grandes grupos, las funciones de cadena y las numéricas. De momento nos ocuparemos de las primeras, que nos permitirán ampliar conocimientos dentro del ámbito del manejo avanzado de cadenas de caracteres, pasando en páginas posteriores al estudio de las funciones de argumento numérico.

Las funciones a que nos referimos en este capítulo son:

STR\$, VAL\$, VAL, y LEN

Las funciones deben operar sobre el tipo de argumento adecuado



La ejecución correcta de una función es FUNCION + ARGUMENTO



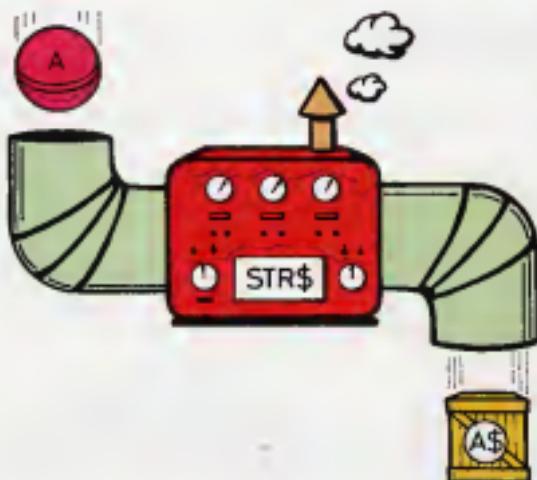
Los argumentos especificados para las diferentes funciones, deben ser coherentes con la función escogida



Las funciones destinadas al tratamiento de cadenas en el BASIC son STR\$, VAL\$, VAL y LEN

Las funciones pueden admitir como argumento cualquier expresión, siempre que sea del tipo adecuado, de cadena, en este caso. Esto quiere decir que es posible adoptar desde la forma más sencilla, es decir, una constante, hasta nombres de variables o expresiones más complejas por combinación de las anteriores, con inclusión opcional de operadores matemáticos y de otro tipo, o incluyendo otras funciones.

Una parte muy importante de la «inteligencia» de nuestro Spectrum (todo aquello que ya sabe des-



La función STR\$ permite convertir las expresiones numéricas en cadenas.

de que encendemos la máquina, sin que nosotros se los expliquemos), se emplea en facilitar esta evaluación de expresiones, que nos permite, como ya hemos visto, utilizar expresiones auténticamente complejas en prácticamente cualquier punto del programa, y con casi todas las sentencias. Esto, que a los recién llegados a la informática nos parecerá tan natural, no lo es tanto, y se da el paradoja de que versiones del lenguaje BASIC para equipos con un costo más de diez veces superior al del Spectrum, carecen de esta gran facilidad para la utilización de expresiones. Para correr este riesgo, brindámos por tanto una cortada avivación a nuestro Spectrum, y más concretamente al sistema que nos proporciona una comodidad tan considerable en el manejo de expresiones. EL EVALUADOR. Volvemos ahora al tema que nos ocupa.

LA FUNCION STR\$

Esta función permite obtener la forma de cadena de una expresión numérica. En cierta manera, es como convertir una expresión de un tipo al otro, lo cual, aunque a primera vista pueda parecer poco trascendente, es de la mayor importancia, debido a que algunas funciones, como ya hemos comentado, sólo son accesibles para determinados tipos de variables (o expresiones), y gracias a esta circunstancia lo son para todas ellas, previa conversión de la expresión al tipo adecuado para la función.

La impresión de variables o expresiones numéricas BASIC, puede representar un problema cuando lo que se desea es que aparezcan en la pantalla «columnas» de números. Esto es debido a que la sentencia PRINT imprime las variables, tanto numéricas como de cadena de izquierda a derecha, mientras que nosotros estamos más acostumbrados a que las expresiones con texto se justifiquen (columnan) por la izquierda y las numéricas por la derecha.

La solución a este problema es sencilla, y se basa en el empleo de la función STR\$, en combinación con LEN y TAB (función que estudiaremos más adelante). El motivo de calcular el equivalente en expresión de cadena de una expresión numérica es, precisamente, poder averiguar su longitud, ya que el BASIC no permite conocer la longitud de una variable o expresión numérica de forma directa, sin su conversión previa a expresión de cadena mediante la función STR\$.

Este hecho nos puede parecer una considerable incomodidad, pero sin embargo se soporta en una sólida base informática, concretamente, en la forma en que la máquina trabaja y almacena la información numérica. Actualmente no estamos capacitados para comprender el por qué de esto, pero antes de terminar la lectura de nuestra obra conoceremos estas intuiciones del BASIC y de la máquina. Por el momento, debe bastarnos con saber que esta regla del BASIC (pase obligatoriamente cualquier dato numérico a cadena para poder conocer su longitud) contribuirá, al igual que la «rigidez» de otras muchas normas del BASIC, a ordenar nuestro esquema mental y cortar en seco nacimientos lo que podrían llegar a ser grandes «vicios» de programación en un futuro no muy lejano.



LEN

La misión de la función LEN es medir la longitud en caracteres de las cadenas.

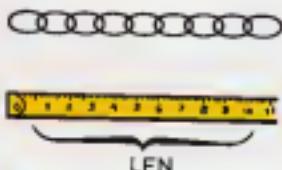


LA FUNCION LEN

INCORRECTO



CORRECTO



A través de la función **LEN**, podemos calcular el número de caracteres contenidos en una variable o expresión de cadena. En caso de que deseemos conocer el número de caracteres que ocuparía la impresión de una variable o expresión de tipo numérico, deberemos calcular primero la **STR\$** de la expresión (**STR\$** es abreviatura de **STRing**, que como ya sabremos, en inglés significa cadena) para, posteriormente, hallar su **LEN** (abreviatura de **LENgth**, que significa longitud). Vamos a ver más claro el uso de **STR\$** y **LEN** con algunos ejemplos.

```

10 REM - STR$ Y LEN
20 LET A= "1234"
30 LET B= "567890"
40 LET AS= "ABCDE"
50 LET BS= "FGHIJKL"
60CLS
70 PRINT "Cadena:";STR$(A); "Longi-
tud:";LEN STR$(A)
80 PRINT "Cadena:";STR$( B); "Longi-
tud:";LEN STR$( B)
90 PRINT "Cadena:";STR$( AS+B); "Longi-
tud:";LEN STR$( AS+B)
100 PRINT "Cadena:";AS;"Longitud:";LEN
A6
110 PRINT "Cadena:";BS;"Longitud:";LEN
B5
120 PRINT "Cadena:";AS+BS;"Longi-
tud:";LEN (AS+BS)

```

Por todos estos motivos, que más adelante comprendremos, es más posible manejar directamente la longitud de un valor numérico.

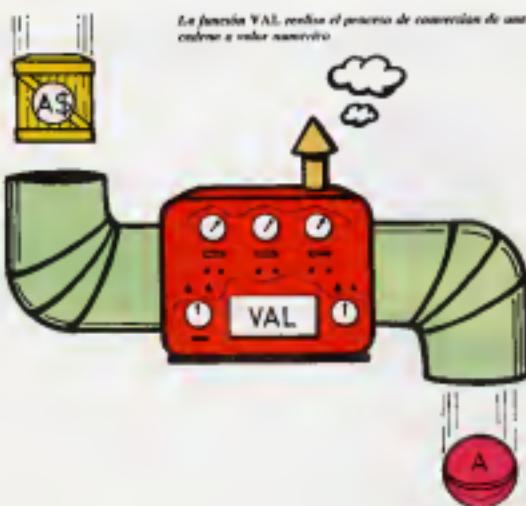
LA FUNCION VAL

La función **VAL** es, ciertamente, la inversa de la función **STR\$**, puesto que calcula el valor numérico de una variable o expresión de cadena. **VAL** es una función de gran potencia, que permite evaluar el contenido de la expresión de cadena (**STR\$**) algo que yo es cadena, habría poco mérito en ello, ¿verdad?



La función **LEN** nos permite calcular la longitud (número de caracteres) de que se compone una cadena, o una expresión de cadena como lo son **A\$+B\$** o **STR\$**. De lo dicho se comprende que el estúdio syntax checker (comprobador de sintaxis), rechazaría claramente cualquiera de nuestros intentos por realizar el **STR\$** de una expresión de cadena, cosa por otra parte bastante lógica, puesto que no hay que pensárselo mucho para darse cuenta de la imposibilidad de convertir en cadena (**STR\$**) algo que yo es cadena, habría poco mérito en ello, ¿verdad?

Gracias a la función **VAL**, podemos emplear nuestro Spectrum como una potente calculadora en la representación de funciones:



La función VAL, realiza el proceso de conversión de una cadena a valor numérico.

no para calcular su valor numérico. Lo veremos más claro con algunos ejemplos:

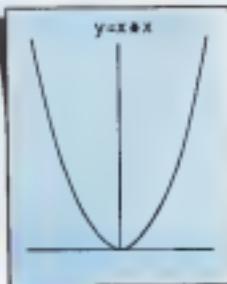
```

10 REM - VAL
20 LET A=1234
30 LET B=567890
40 LET X$="A+B"
50 CLS
60 PRINT X$,VAL X$
```

Hemos comprobado hasta qué punto es capaz la función VAL de indagar en el interior de una cadena de caracteres. Todos los elementos que se

A partir de la tabla de valores que se calcula en el programa de ejemplo, podemos realizar la representación de la función correspondiente.

X	Y
-10	100
-9	81
-8	64
-7	49
-6	36
-5	25
-4	16
-3	9
-2	4
-1	1
0	0
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100



encuentran dentro de ella, tanto constantes literales como variables y operadores aritméticos, son evaluados para dar como resultado un valor numérico.

A pesar de ello, el caso más común en que se utiliza la expresión VAL, es para calcular el valor numérico de una cadena de caracteres, todos ellos numéricos (justo lo inverso a STR\$).

Sólo con un poquito de imaginación, y habiendo observado la extraordinaria capacidad de análisis de la función VAL, debemos intuir que pueden obtenerse los diferentes valores de una función (tabla de valores), que nos permiten dibujar su gráfico. Al fin y al cabo, dada una función matemática cualquiera, por ejemplo $Y=X^2$, hallar la tabla de valores correspondientes para su posterior representación, se limita a ir pidiendo valores de X para obtener valores de Y ; por tanto, ¿qué mejor que la función VAL para cumplir ese cometido? Veámoslo actuando en el ejemplo propuesto:

```

10 REM - TABLA DE VALORES
20 CLS:PRINT "VALOR DE Y","VALOR DE X"
30 LET X$="X^2"
40 FOR X=-10 TO 10
50 PRINT X,VAL X$"
60 NEXT X
```

Esta facilidad en la evaluación de cadenas, que tampoco es frecuente entre otros BASICs, capacita sólo de hallar el VAL de constantes, y no de variables o de expresiones, se la debemos agradecer nuevamente al EVALUADOR, puesto que nos proporciona un sistema magnífico para hallar tablas de valores, etc... Sólo tenemos que efectuar por INPUT la entrada de la variable de cadena a examinar, y luego VAL se ocupará del resto. Así pues, vemos a continuación la línea 30 del mismo ejemplo, que asigna a la cadena X\$ la función a evaluar, por 30 INPUT "Y= ",X\$, lo que nos va a permitir introducir varias funciones sin necesidad de tener que cambiar cada vez el programa.

Pero esta facilidad se puede volver en contra nuestra. Si forcejamos a calcular el valor numérico de una expresión de cadena en la que está contenido algún carácter no numérico, el Spectrum interpreta que se trata de una variable, y busca en la memoria su valor para reemplazarlo en la expresión. En caso de no encontrarlo, el programa se detiene con el mensaje de error 2 [Variable not found (variable no encontrada)]. Esto sucederá en el ejemplo siempre que en la expresión a evaluar introduzcamos variables distintas a X, ya que ésta es la que se utiliza en el bucle de las líneas 40 a 60 para buscar los valores de Y.

Otro de los problemas a los que nos puede conducir el uso de VAL, es a errores del tipo C [Non-



sense in BASIC, al intentar evaluar cadenas que no tienen sentido numérico posible, así por ejemplo, un error de este tipo se producirá al ejecutar **PRINT VAL "3??"**, puesto que **3??** no es una expresión numérica válida. Tenemos en cuenta, que aunque **3** es una constante numérica y el asterisco (*****) es una operación matemática admisible (multiplicación), el signo de interrogación (**?**) no puede ser considerado ni tan siquiera como parte de una variable.

Otro de los problemas que nos pueden surgir con la función que estamos estudiando, son los derivados de la obtención de resultados demasiado grandes o demasiado pequeños, lo que genera errores del tipo **6 Number too big** (número demasiado grande). De entre los problemas de este tipo, probablemente el más frecuente sea el de intentar efectuar divisiones por cero. En nuestro ejemplo, esto puede ocurrir si introducimos la función **VAL(X/X)**, puesto que en el bucle en el que calcula la tabla de valores, la variable **X** recibe desde -10 hasta 10, con un incrementounitario, pasando por tanto por cero.

El evaluador va algo más lejos que lo expuesto hasta el momento, y no capaz de obtener los valores numéricos de expresiones de cadena que no corresponden exactamente en su formato a una expresión de este tipo. Veámoslo con unos ejemplos, en el formato número estándar

los signos aparecen inmediatamente delante del número, del mismo modo, los ceros al comienzo de una cifra nunca se representan.

Una característica de la función **VAL** es la de «normalizar» las expresiones de cadena con el formato estándar de una variable numérica común, para ello, ignora los espacios en blanco que puedan existir entre los elementos de cualquier expresión numérica, así como los ceros a la izquierda de una cifra. Esto nos permite poder efectuar cálculos con variables de cadena, que están preparadas para ser impresas, sin adaptarse concretamente al formato numérico, por ejemplo, con blancos o ceros a la izquierda, sin que ello produzca ningún error. Por otra parte, podemos emplear la función **VAL** para eliminar este tipo de diferencias respecto al formato estándar

BITS

La función **VAL** tiene el sentido inverso al cumplido por **STR\$** sirviéndole el valor numérico de una variable o expresión de cadena.



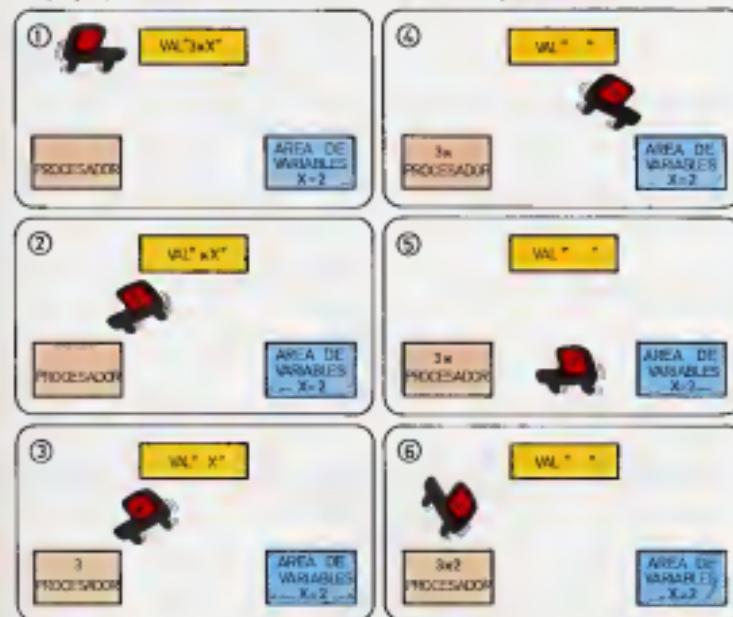
Una característica de la función **VAL**, es la de «normalizar» las expresiones de cadena con el formato numérico estándar.

10 REM - EJEMPLOS DE VAL

20 CLS

30 LET X\$="- 1234" : PRINT X\$,VAL X\$
40 LET X\$="-001234" : PRINT X\$,VAL X\$
50 LET X\$="0001234" : PRINT X\$,VAL X\$
60 LET X\$="- 1234" : PRINT X\$,VAL X\$
70 LET X\$="- 1234" : PRINT X\$,VAL X\$

Diagrama del proceso seguido por el EVALUADOR para el cálculo de la expresión de cadena "X\$".



VAL ignora los espacios en blanco que pueden existir entre los elementos de cualquier expresión numérica, así como los ceros a la izquierda de una cifra.



La función **VAL\$** elimina las dobles comillas de un literal.

VAL "  " = 7

||

VAL "  " = 7

La función VAL ignora los espacios en blanco, entre los elementos de una expresión numérica dentro de una cadena.

i!

En este programa se escribirán en la parte izquierda de la pantalla, formas muy diferentes que pueden adoptar una cifra, y a partir del centro de la misma, la forma normalizada que se consigue mediante el uso de VAL.

LA FUNCION VAL\$

Las funciones pueden admitir como argumento cualquier expresión, siempre que sea del tipo adecuado.



La función STR\$ permite obtener la forma de cadena de una expresión numérica.



Mientras la función LEN, permite calcular el número de caracteres contenidos en una variable de cadena o expresión del mismo tipo.

Antes de entrar en la descripción de sus posibilidades, debemos contar algo acerca de las cadenas de caracteres.

Ya que la asignación de valores a una variable de cadena se hace por medio de una secuencia de caracteres entre comillas, resulta imprescindible poder incluir este último carácter como uno más

La función VAL\$ es capaz de las comillas dobles incluidas dentro de un literal.

dentro de la secuencia que compone la string (Texto, en cambio, un método que permite incluir este carácter dentro de la secuencia, que consiste en teclear una doble comilla en el lugar deseado, además de las comillas simples habituales de principio y fin del literal. Este es un ejemplo que muestra cómo incluir una palabra «entre-comilladas», dentro de las comillas propias de un literal, asignado a una variable:

LET X\$=“ESTA ES UNA PALABRA ““ENTRE-COMILLADA”“ ENTRE COMILLAS”“

La función VAL\$, todo hay que decirlo, es exclusiva del BASIC Sinclair y de muy poco uso. Su misión es eliminar las comillas dobles de una cadena de caracteres, que han de comenzar y terminar por un número impar de comillas, para que sea admitido por el comprobador de sintaxis. Puesto que las comillas dentro de un literal han de ser dobles, los posibles literales sobre los que actúa correctamente VAL\$ son aquellos que comienzan por un número de comillas a partir de tres y contiendo de cuatro en cuatro: 3, 7, 11, 15, etc... De no ser así, o bien la entrada es rechazada por el sufíxus checker (número par de comillas) o bien se produce un error del tipo C Nonsense en BASIC durante la ejecución de la función.

10 REM - USO DE VAL\$
 20 PRINT VAL\$ """SPECTRUM"""
 30 PRINT VAL\$ ""-----SPEC-
 TRUM-----"





En el ejemplo de la línea 20 obtendremos la palabra sin comillas alguna y en el de la 30 con comillas dobles.

Como podemos observar, cualquier literal debe ir encerrado entre comillas y por tanto, cualquier intento de desprecionar de éstas será rechazado por el ordenador. Así pues, pretender emplear la función **VAL\$** sobre un intervalo sin comillas dobles, acabaría a un error del tipo C Nonsense in BASIC, saldría el caso de **PRINT VAL\$ "SPECTRUM"**.

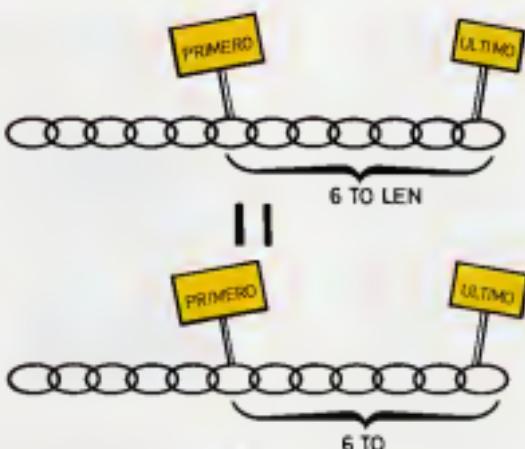
UN PEQUEÑO REPASO

Con todos los conocimientos que hemos obtenido hasta el momento, ya estamos capacitados para llevar a cabo un ejercicio de relativa complejidad, basado en el tratamiento de cadenas realizaremos la tarea de 10 nombres y los presentaremos en la pantalla separándolos por cada espacio en blanco que encontremos. Por ejemplo, la entrada **ANTONIO RODRIGUEZ MENENDEZ**, sería escrita:

**ANTONIO
RODRIGUEZ
MENENDEZ**

Para separar cada grupo de nombres deberemos una líneas en blanco. Ahora, como es habitual, haremos notar una serie de circunstancias que nos pueden ayudar en la confección del programa. En primer lugar, queda claro que la estructura principal del mismo, es un bucle que realiza la siguiente tarea: toma el nombre y lo separa para presentarlo tal como se requiere y, finalmente, deja el espacio en blanco para diferenciar el próximo nombre.

En cuanto a la separación del nombre en subcadena, aprovecharímos nuestros conocimientos recién adquiridos sobre la función **LEN** para completar algo de lo dicho en el capítulo anterior sobre el traccionamiento de strings; entonces vamos cómo tomar la parte derecha de una cadena (por ejemplo **A\$**) hasta el final de la misma se efectúa de la siguiente manera: **A\$ | COMIENZO TO**. Ahora podemos añadir que esta expresión es equivalente a **A\$ (COMIENZO TO LEN A\$)**. De este se infiere que una pista acerca de nuestra manera de buscar los espacios en blanco para separar las partes del nombre entrado, puede ser



Las expresiones que se muestran en el dibujo son equivalentes.

utilizar un bucle cuyo punto inicial sea 1 (primer elemento de la cadena) y cuya final sea la **LEN** de la cadena. Podemos memorizar en una variable el número de carácter en el que se encuentra el último blanco detectado, para así tomar lo siguiente subcadena desde este punto hasta el próximo blanco.

Aunque no está directamente relacionado con el programa que proponemos, aprovecharemos la ocasión para destacar que en ciertos casos es útil conocer los *n* últimos caracteres de una cadena. A tal fin emplearemos la función **LEN**, operando bajo la siguiente fórmula general: *n* últimos caracteres = **A\$ (LEN A\$ - n - 1 TO)**. Así, por ejemplo, si deseamos conocer los cuatro últimos caracteres de la cadena **A\$**, deberíamos escribir **PRINT**

La normalización del formato muestra que VAL lleva a cabo, incluso la impresión de los ceros innecesarios.

$$\text{VAL } "0\ 0\ 0\ 3\ *\ 2" = 6$$



$$\text{VAL } "3\ *\ 2" = 6$$

!

Para saber qué tipo de resultado brinda una función (número o de cadena), basta con observar si el nombre de la función tiene como último carácter un símbolo dólar (\$) o no (resultado numérico).



Los posibles bloques sobre los que se inicia correctamente **VALS**, son aquellos que comienzan y terminan por un número de comillas a partir de tres y contando de cuatro en cuatro (3, 7, 11, 15...).



La expresión **VARIABLE\$([COMIENZO TO])** es equivalente a **VARIABLE\$([COMIENZO TO LEN(VARIABLE\$))**.



Para borrar los n espacios de una cadena (generada a \$), se emplea **A\$([LEN(A\$)-n TO])**.

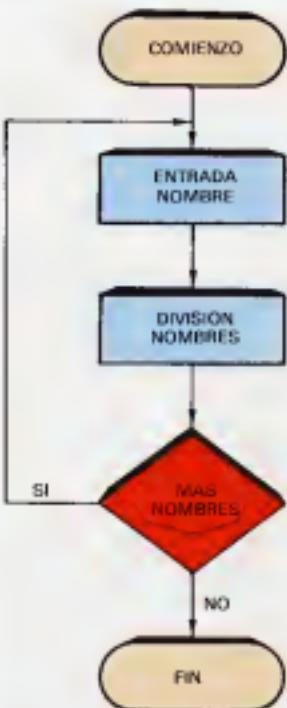


Diagrama de flujo del programa "DIVISION DE NOMBRES".

La función **LLEN** se puede emplear para realizar un círculo inverso, es decir, de derecha a izquierda de una cadena.

A\$([LEN A\$-3 TO]). Por supuesto, se generaría un error en caso de que la cadena A\$ contenga menos de cuatro caracteres.

Bien, ha llegado el momento de que intentemos hacer nuestro propio programa.

```

10 REM DIVISION DE NOMBRES
20 CLS
30 FOR I=0 TO 9
40 INPUT "NOMBRE?";NS: LET P=1
50 FOR J=1 TO LEN NS
60 IF NS$J=" " THEN PRINT NS$P TO J
LET P=J+1
70 NEXT J
80 PRINT
90 NEXT I
  
```

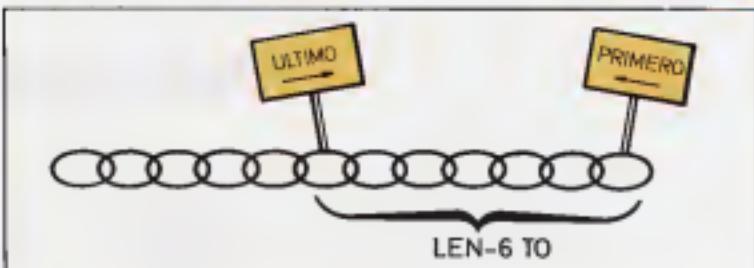
Las líneas 10 y 20 forman la zona de inicializaciones, donde se da título al programa y se borra la pantalla para la presentación más clara de los resultados.

En las líneas 30 y 90 se configura la estructura de bucle basada en la variable I, que consigue que el bloque comprendido entre la línea 40 y la 80 sea repetido 10 veces (de 0 a 9).

En la zona de programa que comprende de la línea 40 a la 80, se realiza el tratamiento de cada nombre. Este comienza por la entrada del mismo a la variable NS mediante un INPUT (línea 40) y finaliza con la impresión de la línea de separación para preparar el tratamiento de un nuevo nombre (línea 80).

Por tanto, la parte más importante de nuestro programa, la cual realiza la división del nombre y la impresión de sus diversas partes, se encuentra en el bucle anidado desde las líneas 50 a la 70. Este bucle de segundo nivel, controlado por la variable J, recorre la cadena entrada (NS) buscando un espacio en blanco (línea 60). Si no lo localiza en el carácter estudiado, continúa analizando el siguiente carácter, y así sucesivamente hasta el final de la cadena.

Si por el contrario localiza un espacio en blanco, escribirá la parte de cadena analizada hasta el momento y hace que la variable P indique el último punto tomado como blanco.



SISTEMAS DE NUMERACION



UALQUIER proceso informático, es decir, de tratamiento de la información, consta básicamente de tres etapas: entrada de los datos, proceso de los mismos y representación de los resultados obtenidos.

Los datos pueden ir desde algo tan simple como dos controles a sumar, hasta algo tan complicado como la información obtenida a partir de sensores que miden la humedad del aire, la temperatura, la presión atmosférica, etc., con el objetivo de establecer un pronóstico del tiempo. El proceso consistirá en aquello semejante de operaciones que han de ejecutarse, para poder obtener un resultado adecuado a nuestras necesidades.

Al igual que los datos, los resultados podrán ser algo sencillo, como la cantidad suma de los dos anteriores, o algo complicado, como un gráfico de líneas de alta y baja presión, con anticipaciones y borrascas, para pronosticar el tiempo de mañana o las próximas semanas.

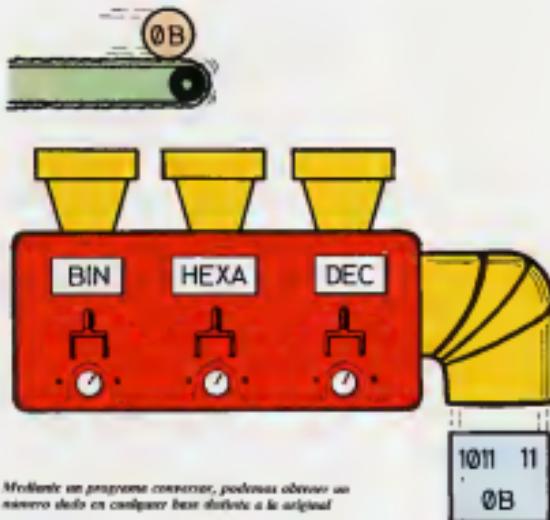
Una mirada a nuestro Spectrum, puede sugerirnos la pregunta de cómo un aparato tan pequeño no capaz de ejecutar infinitud de labores, tan distintas unas de otras. Todo lo anterior, conduce a la idea de ordenador serio e insigüamente inteligente, capaz de «pensar» por sí mismo. Es más, hasta hace no mucho tiempo, los ordenadores eran denominados cerebros electrónicos.

En realidad, esto no es así. Ni el Spectrum pensará como nosotros, ni es tan inteligente como parece. No es capaz de comprender nuestro lenguaje, y las órdenes que le mandamos ejecutar, antes de almacenarlas en su memoria debe traducirlas a un alfabeto propio, formado únicamente por números.

Ahora bien, nosotros estamos acostumbrados a manejar cantidades numéricas en base 10; es decir, para representar un determinado guarismo, formamos una combinación con las cifras que van del 0 al 9. Ni siquiera eso es clípsis de reconocer nuestro micro.

A nadie se le oculta que la electricidad es lo que da vida a nuestro ordenador. Y es aquí donde reside el secreto. Cuando el Spectrum trata de acceder a cualquier información almacenada en la memoria, su particular forma de pensar, sólo sirve para reconocer una cosa: si allí hay corriente o no.

Un convenio adoptado en informática, diferencia entre si estos dos posibilidades, asignándoles



Mientras un procesor procesa, podemos observar un número distinto en cualquier hora distinta a la original.

unos valores. Cuando hay corriente, decimos que allí hay un 1. Cuando no la hay, decimos que está a 0. Esto es lo que se denomina lógica binaria, o sistema binario para la representación de la información. Es decir, entre los distintos componentes por los que circula la información dentro de nuestro micro, sólo podemos encontrar dos estados posibles (1 ó 0). Esta es la unidad mínima de información y se denomina BIT (Binary digit, digito binario).

Con un solo bit, no podemos conseguir mucho. Por ello, se agrupan en grupos de ocho para formar unidades superiores llamadas BYTES. Tenemos, por tanto, 2⁸=256, combinaciones diferentes de unos y ceros, que representarán distintos caracteres (letras, números, características especiales o de control, etc.). Estas 256 posibilidades, configuran el alfabeto privado del Spectrum. Se trata del denominado código A.S.C.I.I.

Este código es el más utilizado en la comunicación entre periféricos, así como para la representación de caracteres y datos en la memoria central de la mayoría de microordenadores.



Para utilizar el programa de conversión, deberemos indicar, al medianteamente dentro del número a convertir, la base en que se encuentra. Para ello elegiremos el siguiente conveniente: 0-Binario, 1-Binaria, y 2-Hexadecimal.



Para el cambio de base BINARIO a HEXADECIMAL, podremos emplear una sencilla tabla de conversión:

BIN	HEX	BIN	HEX
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F



El esfuerzo efectuado para mover los tres manivela del dibujo ha sido siempre el mismo (11 vueltas), solo ha cambiado la representación del mismo, según el número de dígitos de cada número (base de numeración).

DE UN SISTEMA A OTRO

Cuando escribimos un número en base 10, todos sabemos que según nos desplazamos de derecha a izquierda, vamos encontrando primero la cifra de las unidades, luego la de las decenas, centenas, unidades de millar, decenas de millar, etc. Esto es, cada posición representa un número que se obtiene de multiplicar 10 elevado al lugar que ocupa (comenzando a contar desde cero y de derecha a izquierda), por la cifra que se encuentra en dicho lugar. La suma de todas estas cantidades, nos da el valor del número en cuestión.



1 BIT

Las unidades más pequeñas de información son: BIT (dígito binario), NYBLE (4 BITS) y BYTE (8 BITS).



1 0 0 1 NYBLE



1 1 0 0 1 0 0 1 BYTE

Con un ejemplo concreto lo veremos más claramente. Supongamos que hemos escrito 32956. La cifra 6, ocupa el primer lugar comenzando por la derecha. Tendremos entonces $6 \cdot 10^0 = 6^1 = 6$. El siguiente dígito es un 5. Por tanto, $5 \cdot 10^1 = 5 \cdot 10 = 50$. El mismo proceso se sigue con los demás: $9 \cdot 10^2 = 9 \cdot 100 = 900$, $2 \cdot 10^3 = 2 \cdot 1000 = 2000$ y, finalmente, $3 \cdot 10^4 = 3 \cdot 10000 = 30000$. Ahora sumaremos las cantidades que hemos ido obteniendo: $6+50+900+2000+30000$ y el resultado es el número que habíamos tomado en el ejemplo, 32956. Fácil, ¿verdad?

Cualquier otro sistema de numeración, sigue el mismo procedimiento. Lo único que cambiará será la base.

Como vimos antes, el sistema binario o en base dos, tan sólo dispone de dos dígitos, el 0 y el 1. Es decir, un número expresado en esta base, estará formado por una combinación de ceros y unos.

Sea, por ejemplo, el número binario 00101011. Recordemos que la base ahí es 2. Por tanto, comenzando por la derecha tenemos: $2^0 \cdot 1 = 1^1 = 1$, $2^{1} \cdot 1 = 2^1 = 2$, $2^{2} \cdot 0 = 4^1 \cdot 0 = 0$, $2^{3} \cdot 1 = 8^1 \cdot 1 = 8$, $2^{4} \cdot 0 = 16^1 \cdot 0 = 0$, $2^{5} \cdot 1 = 32^1 \cdot 1 = 32$, $2^{6} \cdot 0 = 64^1 \cdot 0 = 0$ y, para terminar, $2^{7} \cdot 0 = 128^1 \cdot 0 = 0$. Sumando todos los resultados, $1+8+32$, hallamos que 43 es el correspondiente valor decimal del número binario 00101011. Como vemos, añadir ceros por la izquierda, no altera el valor del número en cuestión.

La conversión binario-decimal, se efectúa en el Spectrum mediante la función BIN. Si ejecutamos la instrucción PRINT BIN 00101011 la respuesta será sin duda, 43.

Para realizar el paso contrario, decimal a binario, se utiliza el método de las divisiones sucesivas, entre 2. Es decir, tomando el número decimal y lo dividimos por dos. Anotaremos el resto obtenido. Si el cociente es igual o mayor de dos, volvemos a dividir. Señalamos, de nuevo, el resto. El proceso continúa hasta que resulte el cociente menor que dos. El equivalente binario se construye tomando estos números de abajo a arriba, comenzando por el último cociente, y anotándolos de izquierda a derecha. El cuadro adjunto muestra gráficamente el proceso seguido.

UNA NUEVA BASE

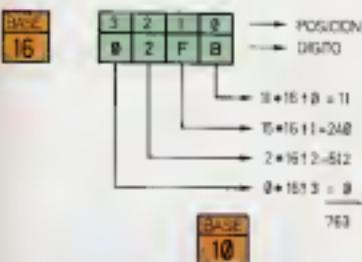
Ocasionalmente, hayamos asistido a errores a los datos de programas escritos en código máquina.

ra, que utilizan una extraña notación, mezcla de números y letras. Y bien, ¿no habíamos quedado en que nuestro ordenador sólo era capaz de identificar dígitos binarios (bits)? Entonces, si el código máquina es lo más parecido al lenguaje de nuestro Spectrum ¿por qué no está formado por unos y ceros?

La respuesta es sencilla. Imaginemos un listado de miles de instrucciones, compuesto únicamente por ceros y unos. ¡A ver quién es el valiente que interprete eso!

Para facilitar la labor de introducción del código máquina, acortando la longitud de sus cifras, se emplea otro sistema de numeración, en el que probablemente hemos visto los listados antes citados: el hexadecimal, que expresa los números en base 16. Mediante este sistema de numeración podemos expresar números más grandes con menor cantidad de dígitos. Además, como

HEXADECIMAL → DECIMAL



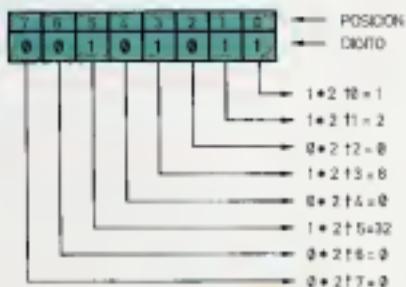
El cambio de base HEXADECIMAL a DECIMAL, se efectúa multiplicando cada dígito por el valor correspondiente que ocupa en la cifra.

Tendremos oportunidad de comprobar más adelante, se muy fácil la conversión del sistema hexadecimal al binario.

El sistema binario dispone de dos dígitos, 0 y 1. El decimal de 10, del 0 al 9. Por tanto, en hexadecimal, tendremos 16 dígitos, que tomarán valores de 0 a 15. Y aquí es donde surge el problema. Estamos hablando de dígitos, es decir, una sola cifra para cada posición dentro del número. De 0 a 9, no hay ninguna ambigüedad, pero para representar más de 10 necesitaremos dos cifras. La cuestión se solventa asignando a los valores que van del 10 al 15 las primeras letras del alfabeto. Según esto, en vez de 10, escribimos A. El 11, es una B, el 12 una C, el 13 una D, el 14 una E y el 15 una F.

La forma de notar los números en base 16, es idéntica a como lo hacímos con los sistemas binario y decimal. Sea, por ejemplo, el número QDFB, en base 16. Tomaremos la primera cifra co-

BINARIO → DECIMAL



43

El cambio de base BINARIO a DECIMAL, se efectúa multiplicando cada dígito por el valor correspondiente que ocupa en la cifra.

BASE
10

BINARIO → HEXADECIMAL

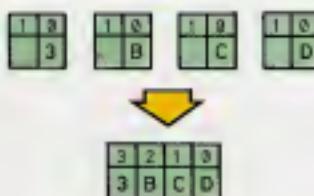
Para el cambio de números en base BINARIO a HEXADECIMAL, resulta útil dividir la cifra completa en grupos de 4 BITS (NÚMEROS), para así poder utilizar una simple tabla de conversión.

BINARIO → HEXADECIMAL



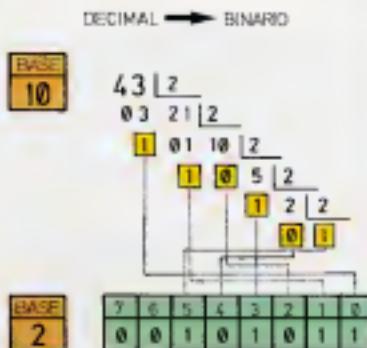
El cambio de base HEXADECIMAL a DECIMAL, se efectúa multiplicando cada dígito por el valor correspondiente que ocupa en la cifra.

TABLA DE CONVERSIÓN

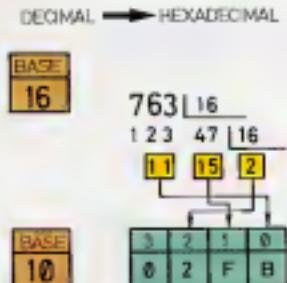


menzando por la derecha, en este caso la B, y vamos construyendo los productos por las potencias de 16: $16^0 \cdot B = 1^{\circ} = 11$. Luego la segunda: $16^1 \cdot F = 16^1 \cdot 240 = 240$. La tercera: $16^2 \cdot 2 = 256 \cdot 2 = 512$. La cuarta es 0, luego no influirá en el resultado. La suma total es $11 + 240 + 512 - 763$. Por tanto, 763 es el valor de canal correspondiente al hexadecimal 02FB.

Cuando convertimos un número expresado en decimal a su notación binaria, utilizaremos el método de las divisiones sucesivas entre dos. La transformación decimal a hexadecimal sigue el mismo procedimiento, sólo que ahora debemos dividir entre 16. Cuando obtengamos un cociente menor que 16, el proceso habrá concluido. Este último cociente o los restos obtenidos, pueden ser expresados a 10. Como sabemos, al este es el



Para la conversión de un número en base DNI-DAT a BINARIO, se emplea el sistema de divisiones sucesivas por dos.



Para la conversión de un número en base DECIMAL a HEXADECIMAL, se emplea el sistema de divisiones sucesivas por decenas.

caso, sustituiremos dichos valores por las letras correspondientes. En el cuadro se ilustra el proceso seguido.

El paso de binario a hexadecimal y viceversa, se efectúa en dos etapas. En la primera convertimos nuestro número a su equivalente decimal, y desde aquí a la base que deseemos, siguiendo los métodos descritos anteriormente. Es conveniente, cuando se trate de números binarios con gran cantidad de cifras, separarlos en grupos de cuatro, comenzando por la derecha. Si el último grupo tiene menos de cuatro dígitos, añadimos el número de ceros necesarios, a la izquierda, hasta completarlo. El ejemplo del cuadro aclaran las ideas.

El proceso de conversión entre los distintos sistemas, realizado a mano, puede resultar largo y tedioso. Nuestro Spectrum, será una gran ayuda. El siguiente programa realiza todo el trabajo.

LA BOMBA

Por una de esas «casualidades» de la vida, nos encontramos cumpliendo nuestro Servicio Militar. Estamos en cuadros en la SERECO (Sección de Reconocimiento), perteneciente a la compañía de armas de apoyo del II Batallón del Regimiento de Infantería más aguerrido de todo la Nación. Aparentemente es un día como otro cualquiera: luce el sol, nos refresca una suave brisa, los hombres del I Batallón padecen a voz en grito «más instrucción». «una jornada muy normal».

De repente, sin apenas darnos cuenta, las sirenas de alarma comienzan aullar, un intenso calor y una breve sensación de ignorancia nos recorre el cuerpo. ¿Qué pasa? Casi al instante empiezan a correr hacia la compañía con todas las fuerzas y velocidad que nos daban nuestras piernas. Los numerosos eran cortos. ¡Los visiones extraterrestres han invadido la Tierra y redondeado nuestro cuartel, situando cargas explosivas en la fachada principal del Regimiento!

«Esto era madez para lo SERECO! El Teniente Kersey y el Sargento Multus, mandos de dicha sección, sujetos por el fango cruzado de las armas tralladoras y subfusiles que con habilidad supieron manejar sus hombres, desatascaron con éxito cada uno de los artificios que los lugartenientes extraterrestres habían colocado.

Perdó... ¿Cómo lo lograron?, ¿qué método utilizaron para desactivar las claves? Eso es precisamente lo que debemos intentar averiguar con este programa.

COMO JUGAR

El Spectrum «piensa» una palabra y nosotros tenemos que adivinarla, introduciendo en cada secuencia del juego una letra. Debemos tener cuidado, porque por cada intento errado, el brazo extensible del detonador se acercará cada vez más a la mecha de la bomba. En el momento que se falle por octava vez, el artefacto quedará irreversiblemente activado y listo para explotar.

Si no hemos conseguido encontrar la clave, nuestro Spectrum hará gala de su caballerizadidad recordándonos la palabra de nuestras desdichas. Como un detalle más del programa, para facilitar nuestra tarea, cada vez que realizamos un intento, la letra utilizada queda impresa en la parte inferior de la pantalla, recordándonos que no debe volver a utilizarse.

EL PROGRAMA



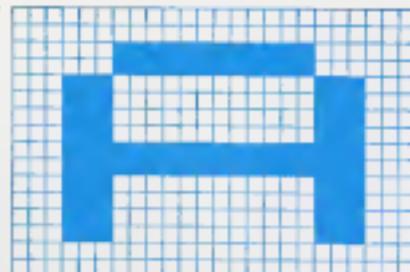
Los caracteres que aparecen con subrayado simple en el teclado, corresponden a los gráficos de los teclados cuya función se subraya.



Los gráficos editados junto con CAPS SHIFT, se representan mediante la tecla correspondiente doblemente subrayada.



PROPORCIONALIDAD
VERTICAL



PROPORCIONALIDAD HORIZONTAL

!

Para la ejecución parcial del código máquina, es imprescindible que el programa sea ejecutado desde la línea 1660.



El código máquina de la subrutina de caracteres gigantes ocupa 277 bytes a partir de la dirección 32266.



El diseño del programa asegura que las palabras programadas por el Spectrum sean escritas al azar y sin repetición. Hasta que el programa no se interrumpe o corgida nuevamente.



El código máquina debe estar situado en la cinta inmediatamente después del final del BASIC.

que aparece incrustado en la bomba, y el resto, el bucle extensible que porta el detonador. Por tanto, conviene recordar que los caracteres que aparecen subrayados en el listado, equivalen a los gráficos de las teclas correspondientes; por otra parte, siguiendo nuestra norma en materia de hábitos, los caracteres doblemente subrayados (sólo podrán ser números), corresponden a los gráficos cambiados de dichas teclas, es decir, **CAPS SHIFT + tecla** (en modo gráfico).

En cuanto al fichero interno que contiene las 50 palabras que puede elegir al azar el Spectrum, puede ser ampliado siempre que en la matriz D definida en la linea 80, realicemos la pertinente actualización, del mismo modo, también será necesario ampliar el rango de elección del RND (línea 660).

Tal y como ha sido diseñado el programa, las palabras se seleccionan aleatoriamente y sin repetición. Debido a esto, puede darse el caso de que al agotarse el banco de palabras, el programa se detenga con el mensaje **E Out of DATA**. En este caso deberemos efectuar **RUN**, para reciclar todo el vocabulario.

A la hora de grabar el programa, lo haremos mediante el uso combinado de los comandos **SAVE** y **LINE** de la siguiente forma:

SAVE "BOMBA" LINE 1650

Para una mayor estética del programa, hemos utilizado la subrutina de caracteres gigantes de PSION Computer, incluida en la cassetta de demostración de los primeros Spectrum. Dicha subrutina deberá ser grabada a continuación del programa BASIC con la dirección 32266 y una longitud de 277 bytes.

Para llevar a cabo esta última parte de la adopción del programa principal, introduciremos en la memoria el siguiente programa:

```
10 REM CARGADOR DE CODIGO MADUINA
20 CLEAR 32266
30 FOR I=32266 TO 32632
40 INPUT A
50 POKE I,A
60 NEXT I
70 PRINT "SITUA LA CINTA AL FINAL DEL
BASIC"
80 SAVE "LIT" CODE 32266,277
```

Este programa realizará la puesta de los 277 números correspondientes al listado de código máquina (introducidosmos por filas); una vez que haya finalizado la entrada de datos, se preparará la grabación del código máquina, para lo cual deberemos situar la cinta inmediatamente a continuación del final del BASIC anteriormente grabado (listado BASIC del programa LA_BOMBA).

```
10 REM *****CINTA DE DATOS*****
20 FOR I=1 TO 50
30 LET A$=RND(1)*255
40 POKE 1650,I
50 NEXT I
60 PRINT "FINISH"
70 END
80 REM *****CINTA DE DATOS*****
90 REM *****CINTA DE DATOS*****
100 REM *****CINTA DE DATOS*****
110 REM *****CINTA DE DATOS*****
120 REM *****CINTA DE DATOS*****
130 REM *****CINTA DE DATOS*****
140 REM *****CINTA DE DATOS*****
150 REM *****CINTA DE DATOS*****
160 REM *****CINTA DE DATOS*****
170 REM *****CINTA DE DATOS*****
180 REM *****CINTA DE DATOS*****
190 REM *****CINTA DE DATOS*****
200 REM *****CINTA DE DATOS*****
210 REM *****CINTA DE DATOS*****
220 REM *****CINTA DE DATOS*****
230 REM *****CINTA DE DATOS*****
240 REM *****CINTA DE DATOS*****
250 REM *****CINTA DE DATOS*****
260 REM *****CINTA DE DATOS*****
270 REM *****CINTA DE DATOS*****
280 REM *****CINTA DE DATOS*****
290 REM *****CINTA DE DATOS*****
300 REM *****CINTA DE DATOS*****
310 REM *****CINTA DE DATOS*****
320 REM *****CINTA DE DATOS*****
330 REM *****CINTA DE DATOS*****
340 REM *****CINTA DE DATOS*****
350 REM *****CINTA DE DATOS*****
360 REM *****CINTA DE DATOS*****
370 REM *****CINTA DE DATOS*****
380 REM *****CINTA DE DATOS*****
390 REM *****CINTA DE DATOS*****
400 REM *****CINTA DE DATOS*****
410 REM *****CINTA DE DATOS*****
420 REM *****CINTA DE DATOS*****
430 REM *****CINTA DE DATOS*****
440 REM *****CINTA DE DATOS*****
450 REM *****CINTA DE DATOS*****
460 REM *****CINTA DE DATOS*****
470 REM *****CINTA DE DATOS*****
480 REM *****CINTA DE DATOS*****
490 REM *****CINTA DE DATOS*****
500 REM *****CINTA DE DATOS*****
```





PROGRAMA

20120414



SENCILLO, ASEQUIBLE, PROFESIONAL

ASÍ ES EL QL DE SINCLAIR, HECHO PÁRA NOSOTROS

Para los profesionales que necesitamos un teclado en nuestro idioma, QL nos ofrece, en castellano, su QWERTY standard de 68 teclas móviles

Para los que deseamos comunicarnos a gran velocidad y capacidad con nuestro ordenador, QL nos presenta su lenguaje SUPER BASIC

Para los que necesitamos gran margen operativo, ahora disponemos de un ordenador con memoria ROM de 32K que contiene el sistema operativo QDOS, un sistema mono-usuario, multi-tarea y con partición de tiempo

Para los que deseamos tener perfectamente ordenada nuestra agenda de trabajo, presupuestos, fichas de productos, nuestra correspondencia, estadísticas de venta, archivo... QL viene dotado de cuatro microdrives totalmente interactivados entre si QL, QUIL, de Tratamiento de

Textos, QL ARCHIVE Base de Datos, QL ABACUS Hoja Electrónica de Cálculo y el QL EASEL para realización de todo tipo de gráficos

Para los que nos gustan las cosas bien acabadas, QL



se suministra con su fuente de alimentación, cables de conexión y adaptadores de TV, monitor y red local, cuatro programas de software de uso genérico, cuatro cartuchos en blanco para los microdrives y manual de instrucciones en castellano

Para los que creemos que lo bien hecho puede tener también el mejor precio, QL ahora por sólo 125.000 pts.

Para los que nos gusta siempre ir bien acompañados, Sinclair —el mayor vendedor del mundo en ordenadores personales— e Investrónica, la mayor red de distribución de España, son nuestras mejores Compañías. Nuestra mejor garantía.

En definitiva, para los que queremos ordenarnos y nunca nos habíamos atrevido.

Con QL ya no hay excusas



DEPARTAMENTO DE VENTAS
 investronica
Tambié Rambla 60 Tel. (34) 407 54 10 34 40 3394 407 54 0990/1000
Calle 38 Tel. (34) 211 26 00 21 12 167 1660/1661